Streamlit Tutorial for Beginners

What is Streamlit?

Streamlit is a Python library that makes it incredibly easy to create interactive web applications for data science and machine learning projects. The beautiful thing about Streamlit is that you don't need to know any HTML, CSS, or JavaScript. You just write Python code, and Streamlit automatically turns it into a web interface.

Think of Streamlit as a way to share your Python scripts with others through a web browser, complete with interactive widgets, charts, and real-time updates.

Step 1: Installation

First, you'll need to install Streamlit. Open your terminal or command prompt and run:

```
pip install streamlit
```

To verify the installation worked, run:

```
streamlit hello
```

This will open a demo application in your browser showing you what Streamlit can do.

Step 2: Your First Streamlit App

Let's create your first app. Create a new file called `app.py` and add this code:

```
import streamlit as st

# This is the simplest Streamlit app possible
st.title("My First Streamlit App")
st.write("Hello, World!")
```

To run this app, open your terminal in the same directory as `app.py` and type:

```
streamlit run app.py
```

Your browser will automatically open with your app running. Notice how `st.write()` can display text, and `st.title()` creates a large heading. The beauty of Streamlit is that every time you save changes to your file, the app automatically updates in your browser.

Step 3: Displaying Different Types of Content

Streamlit has many ways to display content. Let's explore the most common ones:

```
import streamlit as st

# Headers and text
st.title("Content Display Demo")
st.header("This is a header")
st.subheader("This is a subheader")
st.text("This is plain text")
st.write("Write can handle almost anything!")

# Markdown support
st.markdown("### You can use **markdown** too!")

# Code display
st.code("""
def hello():
    print("Hello, Streamlit!")
""", language="python")

# LaTeX for mathematical formulas
st.latex(r"\sum_{i=1}^{n} x_i^2")
```

The `st.write()` function is particularly smart. It automatically detects what type of content you're passing to it and displays it appropriately, whether it's text, dataframes, charts, or other objects.

Step 4: Working with Data

Streamlit makes it easy to display data in tables and dataframes:

```
import streamlit as st
import pandas as pd
import numpy as np

st.title("Data Display Examples")

# Create a simple dataframe
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'Age': [25, 30, 35, 28],
    'City': ['New York', 'Paris', 'London', 'Tokyo']
}
df = pd.DataFrame(data)

# Display as a static table
st.subheader("Static Table")
st.table(df)

# Display as an interactive dataframe
st.subheader("Interactive Dataframe")
```

```
st.dataframe(df)

# You can also highlight specific values
st.subheader("Styled Dataframe")
st.dataframe(df.style.highlight_max(axis=0))

# Display metrics
st.subheader("Metrics")
col1, col2, col3 = st.columns(3)
col1.metric("Temperature", "70 °F", "1.2 °F")
col2.metric("Wind", "9 mph", "-8%")
col3.metric("Humidity", "86%", "4%")
```

Step 5: Creating Visualizations

Streamlit integrates beautifully with popular visualization libraries:

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

st.title("Data Visualization Examples")

# Generate sample data
chart_data = pd.DataFrame(
    np.random.randn(20, 3),
    columns=['A', 'B', 'C']
)

# Native Streamlit charts (simple and fast)
st.subheader("Line Chart")
st.line_chart(chart_data)

st.subheader("Area Chart")
st.area_chart(chart_data)

st.subheader("Bar Chart")
st.bar_chart(chart_data)

# Matplotlib integration
st.subheader("Matplotlib Figure")
fig, ax = plt.subplots()
ax.plot(chart_data['A'], label='Series A')
ax.plot(chart_data['B'], label='Series B')
ax.legend()
st.pyplot(fig)

# Map data (requires latitude and longitude)
st.subheader("Map Visualization")
map_data = pd.DataFrame(
```

```
    np.random.randn(100, 2) / [50, 50] + [37.76, -122.4],
    columns=['lat', 'lon']
)
st.map(map_data)
```

Step 6: Adding Interactivity with Widgets

This is where Streamlit really shines. You can add interactive widgets that let users control your app:

```
import streamlit as st

st.title("Interactive Widgets Demo")

# Text input
name = st.text_input("Enter your name:", "")
if name:
    st.write(f"Hello, {name}!")

# Number input
age = st.number_input("Enter your age:", min_value=0, max_value=120, value=25)
st.write(f"You are {age} years old")

# Slider
temperature = st.slider("Select temperature:", -10, 40, 20)
st.write(f"Current temperature: {temperature}°C")

# Select box
city = st.selectbox(
    "Choose your city:",
    ["New York", "London", "Paris", "Tokyo"]
)
st.write(f"You selected: {city}")

# Multi-select
hobbies = st.multiselect(
    "What are your hobbies?",
    ["Reading", "Sports", "Music", "Cooking", "Travel"]
)
st.write(f"Your hobbies: {', '.join(hobbies)}")

# Checkbox
agree = st.checkbox("I agree to the terms and conditions")
if agree:
    st.success("Thank you for agreeing!")

# Radio buttons
genre = st.radio(
    "What's your favorite movie genre?",
    ["Comedy", "Drama", "Action", "Horror"]
)
st.write(f"You selected: {genre}")
```

```
# Button
if st.button("Click me!"):
    st.balloons()  # Fun animation!
    st.write("Button was clicked!")
```

Step 7: Building a Simple Interactive App

Let's combine what we've learned to create a practical app that calculates compound interest:

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

st.title("💰 Compound Interest Calculator")

st.write("""
This app helps you calculate how your investment will grow over time
with compound interest. Adjust the parameters using the controls below.
""")

# Create input widgets in the sidebar
st.sidebar.header("Investment Parameters")
initial_investment = st.sidebar.number_input(
    "Initial Investment ($)",
    min_value=100,
    max_value=1000000,
    value=10000,
    step=100
)

monthly_contribution = st.sidebar.number_input(
    "Monthly Contribution ($)",
    min_value=0,
    max_value=10000,
    value=500,
    step=50
)

annual_rate = st.sidebar.slider(
    "Annual Interest Rate (%)",
    min_value=0.0,
    max_value=20.0,
    value=7.0,
    step=0.1
)

years = st.sidebar.slider(
    "Investment Period (years)",
    min_value=1,
```

```
    max_value=40,
    value=20
)

# Calculate compound interest
months = years * 12
monthly_rate = annual_rate / 100 / 12

# Build the investment growth array
balance = np.zeros(months + 1)
balance[0] = initial_investment

for month in range(1, months + 1):
    balance[month] = balance[month - 1] * (1 + monthly_rate) +
monthly_contribution

# Calculate totals
total_invested = initial_investment + (monthly_contribution * months)
final_balance = balance[-1]
total_interest = final_balance - total_invested

# Display results
st.header("Results")

col1, col2, col3 = st.columns(3)
col1.metric("Total Invested", f"${total_invested:,.2f}")
col2.metric("Final Balance", f"${final_balance:,.2f}")
col3.metric("Total Interest Earned", f"${total_interest:,.2f}",
            delta=f"{(total_interest/total_invested)*100:.1f}%")

# Create visualization
st.header("Investment Growth Over Time")

fig, ax = plt.subplots(figsize=(10, 6))
time_periods = np.arange(0, months + 1) / 12  # Convert to years

# Calculate principal (money invested)
principal = np.array([initial_investment + (monthly_contribution * month)
                      for month in range(months + 1)])

ax.plot(time_periods, balance, label='Total Balance', linewidth=2, color='green')
ax.plot(time_periods, principal, label='Principal (Invested)',
        linewidth=2, linestyle='--', color='blue')
ax.fill_between(time_periods, principal, balance, alpha=0.3, color='green',
                label='Interest Earned')

ax.set_xlabel('Years')
ax.set_ylabel('Amount ($)')
ax.set_title('Investment Growth with Compound Interest')
ax.legend()
ax.grid(True, alpha=0.3)
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x:,.0f}'))

st.pyplot(fig)
```

```
# Show year-by-year breakdown
if st.checkbox("Show detailed year-by-year breakdown"):
    st.subheader("Annual Breakdown")

    yearly_data = []
    for year in range(years + 1):
        month_index = year * 12
        yearly_data.append({
            'Year': year,
            'Balance': balance[month_index],
            'Total Invested': initial_investment + (monthly_contribution *
month_index),
            'Interest Earned': balance[month_index] - (initial_investment +
(monthly_contribution * month_index))
        })

    df = pd.DataFrame(yearly_data)
    df['Balance'] = df['Balance'].map('${:,.2f}'.format)
    df['Total Invested'] = df['Total Invested'].map('${:,.2f}'.format)
    df['Interest Earned'] = df['Interest Earned'].map('${:,.2f}'.format)

    st.dataframe(df, use_container_width=True)
```

Step 8: Using Columns and Layouts

Streamlit gives you control over your app's layout:

```
import streamlit as st

st.title("Layout Examples")

# Create two columns
col1, col2 = st.columns(2)

with col1:
    st.header("Column 1")
    st.write("This content is in the first column")
    st.button("Button 1")

with col2:
    st.header("Column 2")
    st.write("This content is in the second column")
    st.button("Button 2")

# Create three columns with different widths
st.subheader("Columns with Different Widths")
col1, col2, col3 = st.columns([3, 1, 1])

col1.write("This column is 3x wider")
col2.write("Narrow")
```

```
col3.write("Narrow")

# Expandable sections
with st.expander("Click to expand"):
    st.write("Hidden content that appears when expanded")
    st.image("https://via.placeholder.com/300")

# Containers
container = st.container()
container.write("This is inside a container")

# Sidebar
st.sidebar.title("Sidebar")
st.sidebar.write("You can put widgets here")
sidebar_value = st.sidebar.slider("Sidebar slider", 0, 100, 50)
```

Step 9: Working with File Uploads

Let's create an app that can process uploaded files:

```
import streamlit as st
import pandas as pd

st.title("File Upload Demo")

st.write("Upload a CSV file to see its contents")

uploaded_file = st.file_uploader("Choose a CSV file", type="csv")

if uploaded_file is not None:
    # Read the file
    df = pd.read_csv(uploaded_file)

    st.success("File uploaded successfully!")

    # Show basic information
    st.subheader("Dataset Information")
    st.write(f"Number of rows: {len(df)}")
    st.write(f"Number of columns: {len(df.columns)}")

    # Display the dataframe
    st.subheader("Data Preview")
    st.dataframe(df)

    # Show basic statistics
    if st.checkbox("Show statistical summary"):
        st.subheader("Statistical Summary")
        st.write(df.describe())

    # Let user select columns to visualize
    if len(df.columns) > 0:
```

```
        st.subheader("Visualize Data")
        column = st.selectbox("Select a column to visualize:", df.columns)

        if df[column].dtype in ['int64', 'float64']:
            st.line_chart(df[column])
        else:
            st.write("Selected column is not numeric")
```

Step 10: Adding Session State

Session state allows you to preserve information between reruns of your app:

```
import streamlit as st

st.title("Session State Demo")

# Initialize session state variables
if 'count' not in st.session_state:
    st.session_state.count = 0

if 'name' not in st.session_state:
    st.session_state.name = ''

# Button to increment counter
if st.button('Increment Counter'):
    st.session_state.count += 1

st.write(f'Counter value: {st.session_state.count}')

# Text input that persists
st.session_state.name = st.text_input('Enter your name:', st.session_state.name)
if st.session_state.name:
    st.write(f'Hello, {st.session_state.name}! Your input will persist.')

# Reset button
if st.button('Reset Everything'):
    st.session_state.count = 0
    st.session_state.name = ''
    st.rerun()
```

Step 11: Caching for Performance

When you have expensive computations or data loading, use caching to speed up your app:

```
import streamlit as st
import pandas as pd
import time

st.title("Caching Demo")
```

```
# Without caching - this will run every time the app reruns
def load_data_slow():
    time.sleep(3)  # Simulate slow loading
    return pd.DataFrame({
        'A': range(100),
        'B': range(100, 200)
    })

# With caching - this only runs once
@st.cache_data
def load_data_fast():
    time.sleep(3)  # Simulate slow loading
    return pd.DataFrame({
        'A': range(100),
        'B': range(100, 200)
    })

st.write("Click the button below and notice how the cached version is instant on
subsequent clicks!")

if st.button('Load Data (Cached)'):
    with st.spinner('Loading data...'):
        df = load_data_fast()
    st.success('Done!')
    st.dataframe(df.head())
```

Running Your App

To run any of these examples:

1. Save the code to a file (e.g., `app.py`)
2. Open terminal in that directory
3. Run: `streamlit run app.py`
4. Your browser will open automatically

Tips for Success

Automatic Reloading: Streamlit automatically detects when you save changes to your file and prompts you to rerun the app. You can click "Always rerun" to make this automatic.

Debugging: Use `st.write()` liberally to inspect variables and debug your code. It's like using print statements but with a much better display.

Layout First: When building complex apps, start by planning your layout with columns and containers before adding the logic.

Start Simple: Begin with a basic version of your app and gradually add features. Streamlit makes it easy to iterate quickly.

Explore the Documentation: Streamlit has excellent documentation at https://docs.streamlit.io with many more components and features.

Next Steps

Once you're comfortable with these basics, explore:

- **Plotly** and **Altair** for more interactive charts
- **st.form()** for grouping inputs together
- **Custom components** for extending Streamlit's capabilities
- **Deployment** on Streamlit Cloud, Heroku, or other platforms
- **Multi-page apps** for organizing larger applications

Happy building! Streamlit makes it incredibly easy to turn your Python scripts into shareable web applications. Start with simple apps and gradually build more complex ones as you get comfortable with the framework.